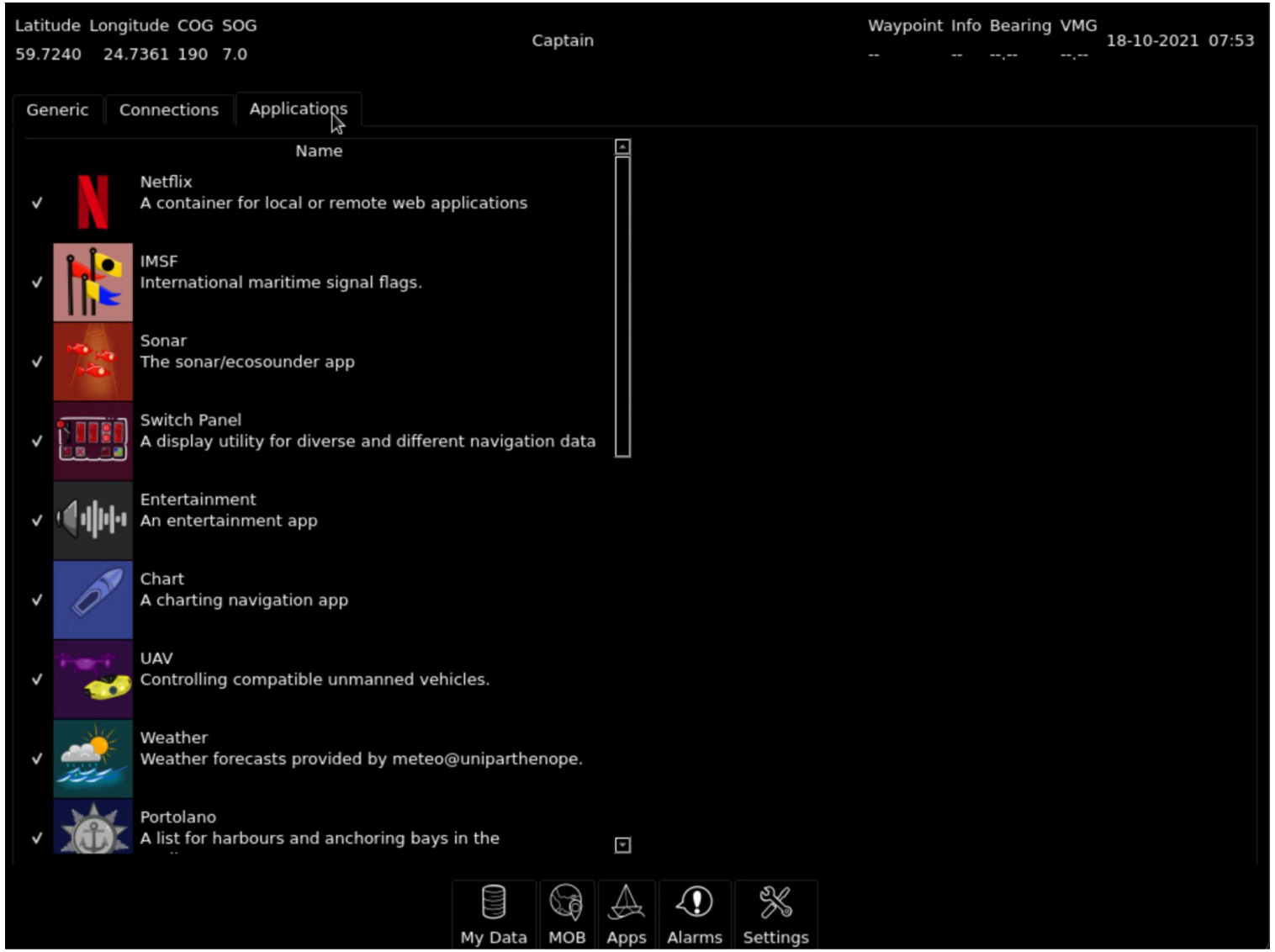
FairWind++

Nuovo modello di impostazioni per le app

Introduzione

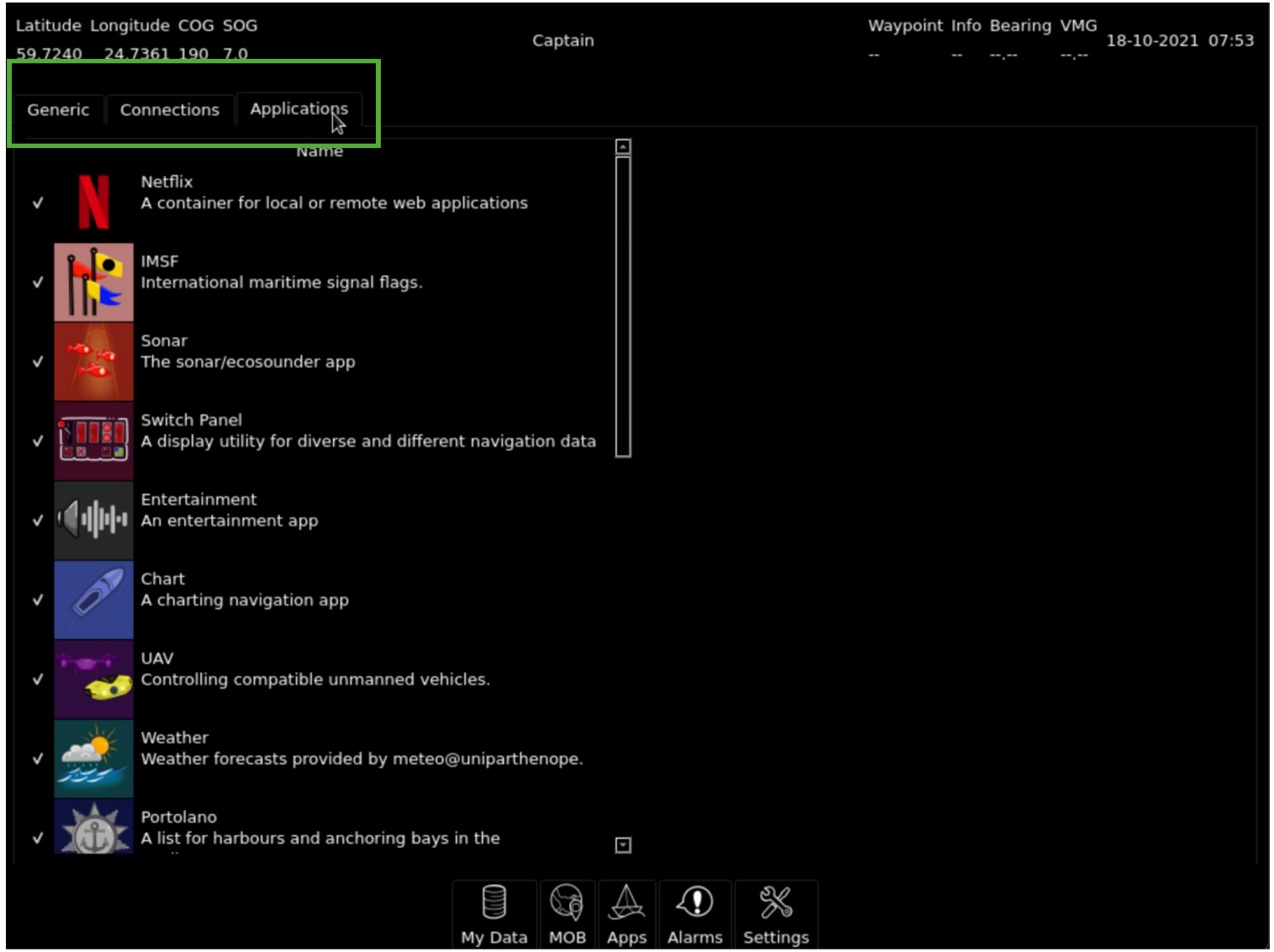
E’ stato richiesto di produrre un nuovo modello per implementare le impostazioni per-app.
Requisiti: semplice, sicuro, espandibile, centralizzato e standard per ogni app.



Situazione attuale

Il modello attuale non rispetta nessuno dei requisiti citati prima.

Al momento, FairWind non dispone di alcun sistema per mostrare le impostazioni dell'applicazione in maniera autonoma, ma chiede all'app stessa di produrle e mostrarle. Non vi è alcun sistema per scrivere ed aggiornare correttamente i valori delle impostazioni. Non c'è neanche uno standard ben preciso da seguire.



```cpp
// Register settings pages inside the FairWind singleton
fairWind->registerSettings(new fairwind::ui::settings::generic::Generic());
fairWind->registerSettings(new fairwind::ui::settings::connections::Connections());
fairWind->registerSettings(new fairwind::ui::settings::applications::Applications());
```

```cpp
class ISettings {
public:
    virtual ~ISettings() = default;

    // Returns the settings page's icon
    virtual QImage getIcon() const = 0;

    // Returns the settings page's name
    virtual QString getName() const = 0;

    // Returns a newly created instance
    virtual ISettings *getNewInstance() = 0;

    // Returns the settings page's class name
    virtual QString getClassName() const = 0;
};
```

```cpp
// Get the singleton instance of FairWind
FairWind *fairWind = FairWind::getInstance();

// Get the registered settings pages from the FairWind instance itself
for (const auto settings: *(fairWind->getSettingsList())) {
    // Add a new tab for each page
    ui->tabWidget->addTab(dynamic_cast<QWidget *>(settings), settings->getName());
}
```

```cpp
// Check if the settings widget has already shown
if (!mSettingsByExtensionId.contains(mExtension)) {

    // Get the extension reference by the application id
    auto extension = fairWind->getAppByExtensionId(mExtension);

    // Get the settings widget
    auto settings = extension->onSettings(nullptr);

    // Store the the settings widget
    mSettingsByExtensionId[mExtension] = settings;
}

// Get the settings widget by the extension id
auto settings = mSettingsByExtensionId[mExtension];

// Set the settings widget in the scroll area
ui->scrollArea_Apps->setWidget(settings);
```

```cpp
QWidget *fairwind::apps::chart::Chart::onSettings(QTabWidget *tabWidget) {
    auto widget = new QWidget();
    auto uiSettings = new Ui::ChartSettings();
    uiSettings->setupUi(widget);

    auto config = getConfig();

    if (config.contains("Options") && config["Options"].isObject()) {
        auto options = config["Options"].toObject();
        if (options.contains("Position") && options["Position"].isString()) {
            uiSettings->lineEditPosition->setText(options["Position"].toString());
        }
        if (options.contains("Heading") && options["Heading"].isString()) {
            uiSettings->lineEditHeading->setText(options["Heading"].toString());
        }

        if (options.contains("Speed") && options["Speed"].isString()) {
            uiSettings->lineEditSpeed->setText(options["Speed"].toString());
        }
    }
    return widget;
}
```

```cpp
QWidget *fairwind::apps::dashboard::Dashboard::onSettings(QTabWidget *tabWidgets) {
    if (m_settings == nullptr) {
        m_settings = new QWidget();
        uiSettings = new Ui::dashboard_Settings();
        uiSettings->setupUi(m_settings);
    }
    return m_settings;
}
```

Proposta

Il nuovo modello deve essere prima di tutto semplice e utilizzabile da qualsiasi app sviluppata per FairWind. Si parte dal JSON Schema, modellato sulla base del file config.json dell'app Chart, essendo il più completo.

Ogni file di configurazione per un'app dovrà fornire due oggetti necessari: Settings e Values. Il primo è un array di impostazioni, l'altra un oggetto che conterrà i valori correnti delle impostazioni.

```
{
   "$schema": "https://json-schema.org/draft/2019-09/schema" ,
   "title": "Chart app JSON Schema" ,
   "type": "object",
   "properties" : {
       "Settings" : {
           "type": "array",
           "items" : [
               {
                   "type": "object",
                   "properties" : {
                       "id": {
                           "type": "string"
                       },
                       "displayName" : {
                           "type": "string"
                       },
                       "widgetClassName" : {
                           "type": "string",
                           "oneOf" : [
                               {
                                   "enum" : [
                                       "fairwind::ui::settings::FairCheckBox" ,
                                       "fairwind::ui::settings::FairComboBox" ,
                                       "fairwind::ui::settings::FairLineEdit"
                                   ]
                               }
                           ]
                       },
                       "defaultValue" : {
                           "type": "string"
                       },
                       "domain" : {
                           "type": "array"
                       }
                   },
                   "required" : [
                       "id",
                       "displayName" ,
                       "widgetClassName" ,
                       "defaultValue"
                   ]
               }
           ]
       },
       "Values" : {
           "type": "object"
       }
   },
   "required" : [
       "Settings",
       "Values"
   ]
}
```

Esempio Chart

L’app Chart, nel suo file di configurazione, fornisce un array di cinque impostazioni, di tipo FairLineEdit, FairCheckBox e FairComboBox.

Nell’oggetto Values, troviamo quindi cinque coppie chiave/valore: la chiave è l’ID dell’impostazione e il valore è esattamente il valore corrente.

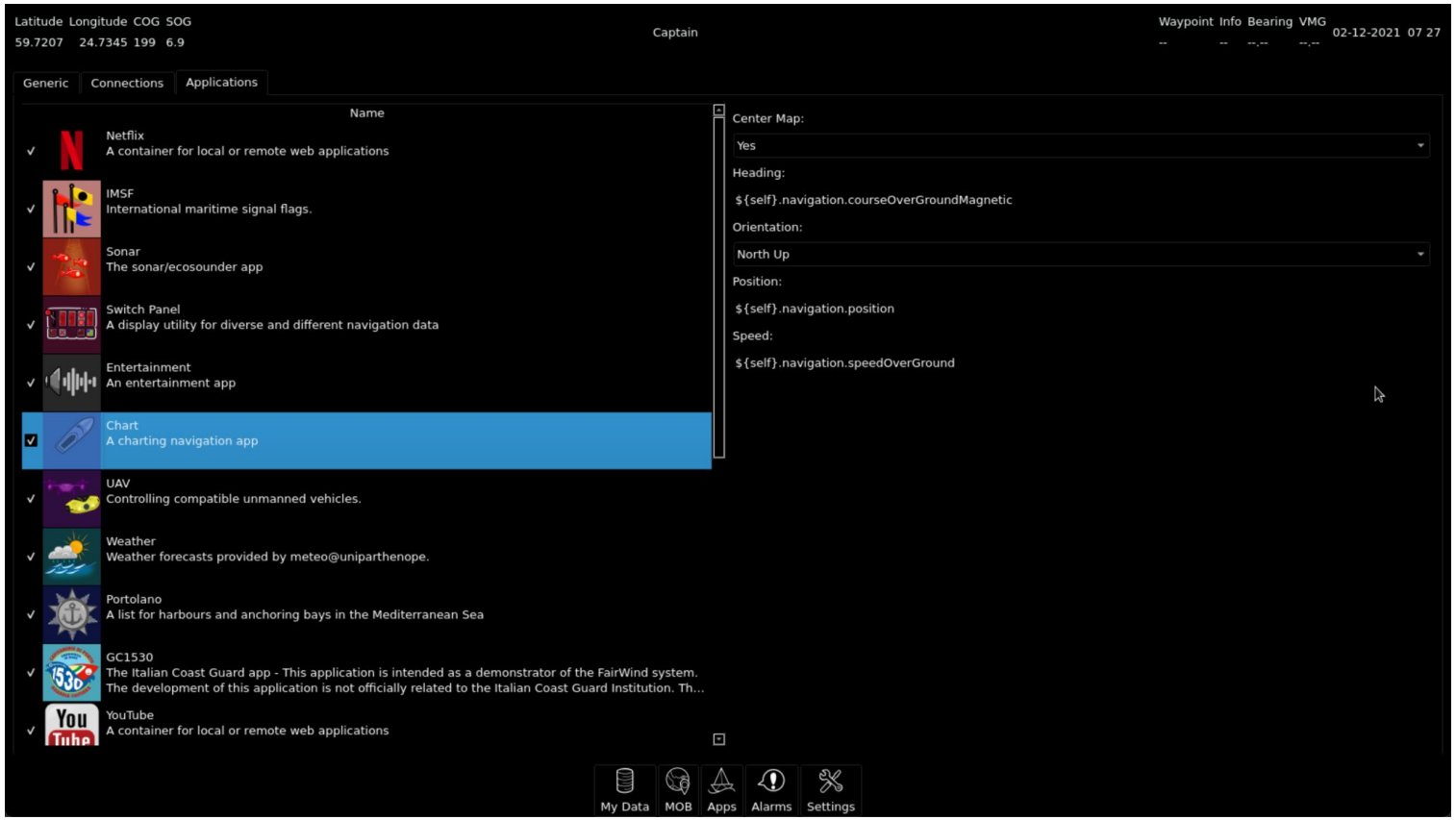
Ogni proprietà è in formato stringa per semplicità di implementazione.


```
"Settings": [
        {
            "defaultValue": "No",
            "displayName": "Center Map",
            "domain": [
                "No",
                "Yes"
            ],
            "id": "chart.settings.center_map",
            "widgetClassName": "fairwind::ui::settings::FairComboBox"
        },
        {
            "defaultValue": "${self}.navigation.courseOverGroundMagnetic",
            "displayName": "Heading",
            "id": "chart.settings.heading",
            "widgetClassName": "fairwind::ui::settings::FairLineEdit"
        },
        {
            "defaultValue": "North Up",
            "displayName": "Orientation",
            "domain": [
                "North Up",
                "Course Up"
            ],
            "id": "chart.settings.orientation",
            "widgetClassName": "fairwind::ui::settings::FairComboBox"
        },
        {
            "defaultValue": "${self}.navigation.position",
            "displayName": "Position",
            "id": "chart.settings.position",
            "widgetClassName": "fairwind::ui::settings::FairLineEdit"
        },
        {
            "defaultValue": "${self}.navigation.speedOverGround",
            "displayName": "Speed",
            "id": "chart.settings.speed",
            "widgetClassName": "fairwind::ui::settings::FairLineEdit"
        }
    ],
    "Values": {
        "chart.settings.center_map": "Yes",
        "chart.settings.heading": "${self}.navigation.courseOverGroundMagnetic",
        "chart.settings.orientation": "North Up",
        "chart.settings.position": "${self}.navigation.position",
        "chart.settings.speed": "${self}.navigation.speedOverGround"
    }
```

E quindi?

L’idea è che sia l’app a fornire le impostazioni al sistema FairWind e ad aggiornarle quando richiesta, dal sistema appunto, ma quando si parla di visualizzarle nell’interfaccia e di gestire l’interazione con l’utente, il responsabile deve essere FairWind.

In questo modo si ha la certezza che un’app non possa utilizzare widget e valori non validi o pericolosi.



```cpp
// Register settings pages inside the FairWind singleton
fairWind->registerSettingsTab(new fairwind::ui::settings::generic::Generic());
fairWind->registerSettingsTab(new fairwind::ui::settings::connections::Connections());
fairWind->registerSettingsTab(new fairwind::ui::settings::applications::Applications());

// Register the settings widgets inside the FairWind singleton
fairWind->registerSettings(new fairwind::ui::settings::FairComboBox());
fairWind->registerSettings(new fairwind::ui::settings::FairLineEdit());
fairWind->registerSettings(new fairwind::ui::settings::FairCheckBox());
```

```cpp
class ISettings{
public:
    virtual ~ISettings() = default;

    /*
     * setDetails
     * This method sets the state of the widget
     */
    virtual void setDetails(QJsonObject settings, QJsonObject values, fairwind::apps::IApp* extension) = 0;

    /*
     * getNewInstance
     * Returns a new instance of ISettings
     */
    virtual ISettings* getNewInstance() = 0;

    /*
     * getClassName
     * Returns the class name of the settings
     */
    virtual QString getClassName() = 0;
};
```

```cpp
void fairwind::apps::chart::Chart::updateSettings(QString settingsID, QString newValue) {
    AppBase::updateSettings(settingsID, newValue);
}

void fairwind::apps::chart::Chart::setConfig(QJsonObject config) {
    AppBase::setConfig(config);
}
```

```cpp
// Get the 'Settings' object from the config
auto settings = configs["Settings"].toArray();
// Get the 'Values' object from the config
auto values = configs["Values"].toObject();

// Iterate on all the extension's settings
for (int i = 0; i < settings.size(); i++) {
    // Generate the widget according to the provided class name
    auto widget = fairWind->instanceSettings(settings[i].toObject()["widgetClassName"].toString());

    // Check if the widget is valid
    if (widget != nullptr) {
        // Create a label
        auto label = new QLabel(settings[i].toObject()["displayName"].toString() + ":");

        // Insert the label
        settingsTable->insertRow(settingsTable->rowCount());
        settingsTable->setCellWidget(settingsTable->rowCount() - 1, 0, label);

        // Set the details for the widget
        widget->setDetails(settings[i].toObject(), values, extension);

        // Add the widget to the container
        settingsTable->insertRow(settingsTable->rowCount());
        settingsTable->setCellWidget(settingsTable->rowCount() - 1, 0, dynamic_cast<QWidget *>(widget));
    }
}

// Set the settings widget in the scroll area
ui->scrollArea_Apps->setWidget(settingsTable);
```

```cpp
/*
 * updateSettings
 * This method will update the app's settings inside its json config file
 * and wil update the m_config variable accordingly
 */
void fairwind::AppBase::updateSettings(QString settingsID, QString newValue) {
    // Get the path
    QDir appDataPath = QDir(getMetaData()["dataRoot"].toString() + QDir::separator() + getId());

    // Create the path if needed
    appDataPath.mkpath(appDataPath.absolutePath());

    // Set the config.json file
    QFile configsFile(appDataPath.absolutePath() + QDir::separator() + "config.json");
    configsFile.open(QFile::ReadWrite);

    // Get config
    QJsonObject configs = getConfig();

    // Find the 'Values' object inside the configs
    QJsonValueRef ref = configs.find("Values").value();
    QJsonObject values = ref.toObject();

    // Update the settings value
    values.insert(settingsID, newValue);

    // Save the changes
    ref = values;

    auto configsDocument = new QJsonDocument;
    configsDocument->setObject(configs);

    // Wipe the config file and then fill it with the new content
    if (configsFile.resize(0))
        configsFile.write(configsDocument->toJson());

    // Close the file
    configsFile.close();
    // Set the config
    setConfig(configs);
}
```

FairLineEdit

```
void fairwind::ui::settings::FairLineEdit::setDetails(QJsonObject settings, QJsonObject values, fairwind::apps::IApp* extension) {
    // Get the settings ID
    auto settingsID = settings["id"].toString();

    // Set the current value
    this->setText(values[settingsID].toString());

    // When the current value changes, call the updateSettings method to save the changes
    connect(this,static_cast<void (QLineEdit::*)(const QString& newValue)>(&QLineEdit::textChanged), this, [settingsID, extension](QString newValue) {
        extension->updateSettings(settingsID, newValue);
    });
}
```

FairCheckBox

```
void fairwind::ui::settings::FairCheckBox::setDetails(QJsonObject settings, QJsonObject values, fairwind::apps::IApp* extension) {
    // Get the settings ID
    auto settingsID = settings["id"].toString();

    // Get the settings current value
    QString checkState = values[settingsID].toString();

    // Set the checkbox's state according to the current value
    if (checkState.toInt() == 0)
        this->setCheckState(Qt::CheckState::Unchecked);
    else
        this->setCheckState(Qt::CheckState::Checked);

    // When the current value changes, call the updateSettings method to save the changes
    connect(this,static_cast<void (QCheckBox::*)(int state)>(&QCheckBox::stateChanged), this, [settingsID, extension, checkState]() {
        extension->updateSettings(settingsID, checkState == "0" ? "2" : "0");
    });
}
```

FairComboBox

```
void fairwind::ui::settings::FairComboBox::setDetails(QJsonObject settings, QJsonObject values, fairwind::apps::IApp* extension) {
    // Get the settings ID
    auto settingsID = settings["id"].toString();

    // Get the settings possible values
    auto domain = settings["domain"].toArray();

    // Add the current value
    this->addItem(values[settingsID].toString());

    // Add the remaining values from the domain
    for (int j = 0; j < domain.size(); j++) {
        if (domain[j].toString() != values[settingsID].toString())
            this->addItem(domain[j].toString());
    }

    // When the current value changes, call the updateSettings method to save the changes
    connect(this,static_cast<void (QComboBox::*)(int index)>(&QComboBox::currentIndexChanged), this, [settingsID, extension, this]() {
        extension->updateSettings(settingsID, this->currentText());
    });
}
```

Grazie